RenPar'20 / SympA'14 / **CFSE 8**
Saint-Malo, France, du 10 au 13 mai 2011

Architectures de filtrage reconfigurables dynamiquement

Mathieu Valero, Luciana Arantes, Maria Gradinariu Potop-Butucaru, Pierre Sens

LIP6 - Université Paris 6 - CNRS - INRIA
4 place Jussieu,
75005 Paris - France
Prénom.Nom@lip6.fr

Résumé
Les R-trees distribués (DR-trees) permettent de mettre en œuvre facilement différents types de systèmes ; de requêtes par intervalles, de publication/abonnement ou de structures k-NN. Ils héritent ce faisant des bonnes propriétés de ce type d'arbre ; hauteur logarithmique, nombre borné de voisins et équilibre structurel global. Cependant la distribution des nœuds du DR-tree sur les pairs n'a pas été suffisament considérée. Jusqu'à présent, elle résulte de l'ordre d'arrivée des pairs et/ou de leurs relations sémantiques. Or en ignorant l'hétérogénéité du réseau, la distribution peut engendrer un écart important entre les performances théoriques et expérimentales du système. Pour pallier ce problème, cet article propose une réification de la distribution et un mécanisme de migration dynamique de noeuds. Celui ci permet de modifier la distribution, donc les performances du système, sans modifier le DR-tree. Nos simulations montrent qu'il permet d'améliorer substantiellement les performances du système.

Mots-clés : P2P, R-tree, reconfiguration, filtrage, migration

1. Introduction

La relation entre les liens logiques et physiques d'un overlay a été amplement étudiée depuis les premiers systèmes P2P ; Pastry [19] et CAN [18] tenaient déja compte de la distance géographique entre les pairs. De nombreux intergiciels P2P définissent des notions de proximité entre les pairs ; en particulier dans les systèmes de publication/abonnement où elle est fonction de la similitude sémantique des abonnements. L'implémentation de ces systèmes peut être optimisée selon différentes métriques comme la latence ou la distribution de charge.

La plupart des architectures utilisent des structures arborescentes qui se prêtent naturellement au filtrage. La conception d'un tel système est soumise à un compromis entre la conservation des propriétés structurelles de l'arbre et l'optimisation des communications en termes de latence et de charge.

Cet article s'inscrit dans cette problématique de recherche. Il traite en particulier des DR-tree où les performances globales du système dépendent grandement de la distribution des nœuds logiques sur les pairs physiques. Nous proposons un mécanisme permettant de modifier dynamiquement la distribution de ces nœuds. Nous tudions galement l'impact de la distribution des nœuds sur les performances du système.

Les DR-tree [2] sont une version distribuée des R-tree [11] qui permettent de gérer des objets représentables par un rectangle multi-dimensionnel. Les DR-tree sont des overlays P2P où chaque pair a un nombre borné de voisins garantissant une complexité logarithmique en nombre de sauts pour les opérations de recherche et de diffusion. En représentant les abonnements par des rectangles et les publications par des points, ils permettent d'implémenter facilement des systèmes de publication/abonnement basés sur le

contenu. Une particularité importante de la conception des DR-tree est qu'un pair physique peut être responsable de plusieurs nœuds logiques de l'arbre. De ce fait si, pour une métrique donnée, un "mauvais" pair est responsable de plusieurs nœuds et/ou de nœuds proches de la racine, il est évident que les performances du système s'en ressentiront.
Notre contribution tend à éviter ce genre de situations, grace à un mécanisme de placement initial des nœuds et grace à un mécanisme de migration dynamique de nœuds. Nous définissons une notion de validité pour les migrations puis nous présentons un protocole décentralisé, inspiré des techniques transactionnelles, permettant d'éviter les conflits lors de reconfigurations concurrentes.

2. Travaux connexes

Le placement des nœuds logiques a été abordé dans plusieurs domaines. Dans les systèmes de publication/abonnement comme Meghdoot [10], Mirinae [7], Rebeca [20], SCRIBE [6] ou Sub-2-Sub [17], chaque nœud logique correspond à un abonnement et chaque pair est responsable de ses propres abonnements. Dans BATON [13] et VBI [14], deux plates-formes d'indexation basées sur des AVL, les nœuds logiques sont divisés en deux catégories ; les feuilles, utilisées pour le stockage, et les nœud internes, utilisés pour le routage. Chaque pair est responsable d'une feuille et éventuellement d'un nœud interne. Dans les DR-tree [2] chaque pair physique est responsable d'une feuille et éventuellement d'une partie de ses ancêtres directs.
De nombreux systèmes de publication/abonnement [3, 1, 8, 5] utilisent des arbres de diffusion optimisés selon une métrique particulière. Notre système n'utilise aucune structure en plus des DR-tree et la métrique d'optimisation est un paramètre de notre mécanisme de migration dynamique. Pour ces raisons notre approche est plus générique.
Brushwood [4] est un overlay basé sur des kd-tree visant à préserver la localité logique des données tout en équilibrant la charge. Chaque pair est responsable d'un nœud logique qui représente un hyperplan k-dimensionnel. Quand un nouveau pair rejoint le système, il est routé vers un pair. Si ce dernier est surchargé, il délgue la moitié de son hyperplan au nouvel arrivant. Brushwood [4] propose de plus un mécanisme d'évaluation sporadique de la charge. Les pairs surchargés peuvent forcer ceux qui ne le sont pas à se réinserer afin de bénéficier à nouveau du protocole d'arrivée. Chordal graph [15] est un système de requêtage par intervalle. A l'instar de SkipNet [12, 9], chaque pair appartient à plusieurs anneaux. Pour chacun d'eux, le pair est responsable d'un nœud logique qui représente un intervalle de valeurs. Chacun de ces intervalles est susceptible d'être redimensionné pour équilibrer la charge.
Chordal graph [15] et Brushwood [4] proposent des mécanisme de répartition de charge qui modifient les nœuds logiques ; par redimensionnement des intervales ou par division des hyperplans. Notre approche modifie la distribution du DR-tree mais ne modifie pas la structure logique.

3. Background

Dans cette section nous rappelons certaines définitions et caractéristiques des DR-tree [2]. Nous présentons également le concept de distribution des noeuds virtuels.

3.1. R-trees distribués

Les R-tree [11] sont des arbres équilibrés permettant de gérer des objets (dits spatiaux) représentables pas un rectangle multi-dimensionnel. Chaque feuille pointe vers un certain nombre d'objets spatiaux. Un R-tree est caractérisé par les propriétés suivantes :
– Si la racine n'est pas une feuille, elle a au moins deux fils,
– Chaque noeud non feuille a au plus M fils,
– Chaque noeud non feuille a au moins m fils (avec $m \leq M/2$),

– Toutes les feuilles sont au même niveau.

Les R-tree distribués (DR-trees [2]) étendent la structure d'indexation des R-tree o les pairs physiques s'organisent en fonction de leurs relations sémantiques. Ils bénéficient de certaines bonnes proproétés des R-tree ; chaque pair physique a un degr borné et le temps de recherche dans la structure est proportionnel au logarithmique du nombre de pairs.

Les machines physiques connectées au système seront appelées *p-nodes* (pour *physical nodes*). Un DR-tree est une structure virtuelle distribuée parmi un ensemble de *p-nodes*. Les nœuds du DR-tree seront donc appelés *v-nodes* (pour *virtual node*). Par la suite, les termes relatifs au DR-tree seront prefixés par "v-". La racine du DR-tree sera ainsi appelée *v-root* tandis que ses feuilles seront appelées *v-leaves*. A part la racine, chaque v-node *n* a un père (*v-father(n)*) et, si ce n'est pas une feuille, un ensemble de fils (*v-children(n)*). Ces nœuds sont les voisins de *n* (*v-neighbors(n)*).

Le graphe de communication physique est défini par la distribution du DR-tree sur les $p-nodes$ participants au système. Intuitivement, il y a un arc entre deux sommets s'ils sont responsables de nœuds logiques voisins. Il y a un $p-edge$ (p,q), $p \neq q$ dans le graphe de communication s'il y a un v-edge (s,t) dans le DR-tree o *p* est responsable de *s* et *q* est responsable de *t*.

La figure 1 montre le DR-tree composé des *v-nodes* $\{n0,\ldots,n12\}$ distribués sur les *p-nodes* $\{p1,\ldots,p9\}$. Les pointillés indiquent comment les *v-nodes* sont distribués. Il y a un arc entre $p1$ et $p5$ dans le graphe de communication car il y a un arc $(n0,n6)$ dans le DR-tree, $p1$ est responsable de $n0$ et $p5$ est responsable de $n6$.

Dans l'implémentation des DR-tree [2], la distribution est déterminée dans les procédures d'arrivée et de départ de *p-nodes*. Quand un *p-node* arrive, il crée un *v-node*, plus prcisment une *v-leaf*. Il contacte alors un autre *p-node* et insère le *v-node* dans la structure existante. Pendant l'insertion, certains autres nœuds logiques peuvent éclater (*split*) afin de préserver les contraintes de degrès des R-tree. De manire symtrique, le retrait d'un *v-node* peut entraner un regroupement de *v-nodes* (*collapse*).

Il résulte de cette implémentation que les propriétés suivantes sont toujours vérifiées :

– $Inv1$: chaque *p-node* est responsable d'une feuille.
– $Inv2$: si un *p-node p* est responsable d'un *v-node n*, soit *n* est une feuille soit *p* est responsable d'exactement un fils de *n*.

Intuitivement chaque *p-node* est responsable d'une chaîne de *v-nodes*. Autrement dit chaque *p-node* est responsable d'une feuille et d'une suite contigüe de ses prédécesseurs. Nous appellerons le *premier* nœud d'un *p-node*, le *v-node* le plus proche de la racine dont il est responsable et le *dernier* nœud d'un *p-node* celui qui en est le plus éloigné. Par exemple dans la figure 1a, $n6$ est le premier nœud de $p5$ et $n7$ son dernier. Les invariants précités assurent que le graphe de communication est un arbre :

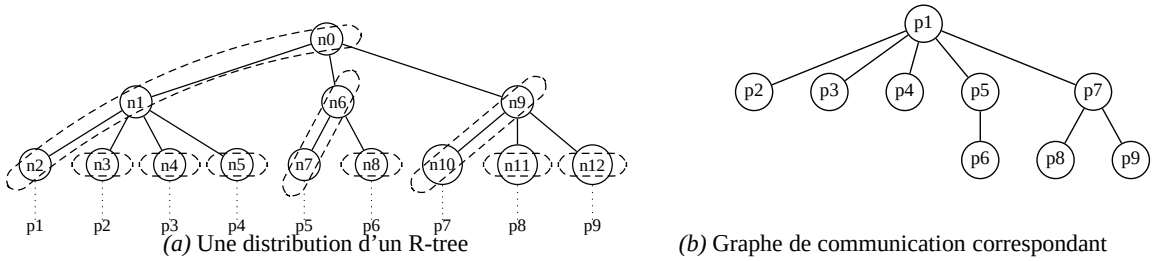
– La racine est le *p-node* responsable de la *v-root*.
– Un *p-node p* est le père d'un *p-node q* si *p* est responsable du père du *premier* nœud de *q*.

Par exemple sur la figure 1a, *p-father(p5) = p-father(p7) = p1*. Les propriétés de la distribution assurent également que chaque pair a un nombre borné de voisins dans le graphe de communication. Dans un système comptant *N* nœuds physiques et avec un DR-tree de degré $[\![m;M]\!]$, la hauteur de l'arbre est de $log_m(N)$; la racine du graphe de communication est donc responsable de $log_m(N)$ nœuds. Comme chaque nœud a au plus *M* voisins, la racine du graphe de communication a au plus $M * log_m(N)$ voisins.

3.2. Distribution des nœuds logiques

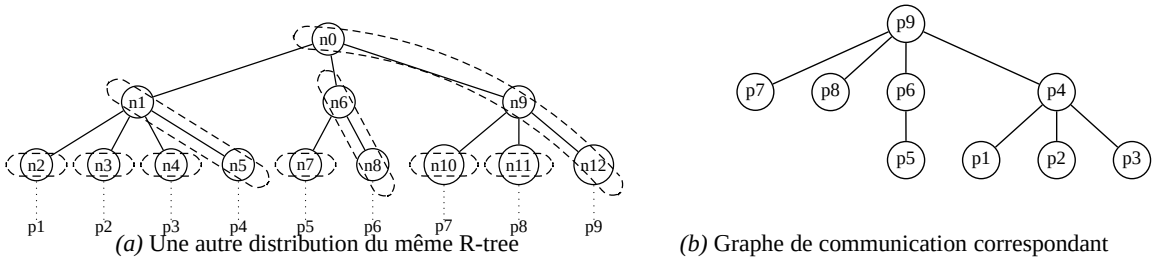
Un DR-tree est une structure logique distribuée parmi un ensemble de machines physiques. Les figures 1 et 2 montrent deux distributions différentes du même R-tree sur un ensemble de *p-nodes* $\{p1,\ldots,p9\}$. La distribution détermine le graphe de communication et a de ce fait un impact fort sur les performances du système.

Dans [2] la distribution dépend de l'ordre d'arrivée des *p-nodes*, de l'implémentation du protocole de *join*



(a) Une distribution d'un R-tree *(b)* Graphe de communication correspondant

FIGURE *1* – Distribution d'un DR-tree et graphe de communication correspondant



(a) Une autre distribution du même R-tree *(b)* Graphe de communication correspondant

FIGURE *2* – Une autre distribution du même DR-tree menant à un graphe de communication différent

et de l'implémentation de la méthode $split$ du DR-tree. Cette approche a deux principaux inconvénients. Premièrement les caractéristiques du réseau ne sont pas prises en compte. Deuxièmement la distribution est statique. Les nœuds sont placés à leur création ou durant les $join/split$ et ils ne sont jamais déplacés. Le premier point est problématique dans les réseaux hétérogènes où les performances du système sont étroitement liées à la capacits des nœuds "proches" de la racine. Par exemple si selon une métrique donnée p_1 est un "bon" pair et p_9 un "mauvais", le graphe de communication de la figure 1a est meilleur que celui de la figure 2a. Le second est problématique si la métrique d'évaluation des performances évolue au cours du temps. Par exemple, les performances du système se dégraderont si la qualité des *p-nodes* responsables des nœuds proches de la racine diminue.

Pour dépasser ces deux limitations, nous proposons un mécanisme de migration dynamique de nœuds virtuels. Il permet de modifier la distribution du DR-tree (sans le modifier lui même) pour améliorer les performances du système et s'adapter à l'évolution de la qualité des pairs. Notre mécanisme utilise les informations d'une fonction de coût (par exemple la charge des nœuds physiques) basée sur les interactions dans le graphe de communication.

4. Mécanisme de migration dynamique

Dans cette section nous définissons la notion de cohérence d'une migration. Nous adoptons une politique conservatrice ; nous souhaitons préserver les invariants de distribution décrits dans la section 3.1. Nous allons donc restreindre les possibilités de migrations de faon à n'obtenir que des distributions qui auraient pu résulter du protocole original des DR-tree [2]. Il faut dès lors définir, à l'échelle d'un *p-node*, quels sont les *v-nodes* candidats à la migration et pour chacun de ces nœuds quelles sont les destinations possibles. Nous présentons ensuite le protocole de migration et étudions quand il est utilisé.

Nous noterons $p \xrightarrow{n} q$ la migration du nœud virtuel n du *p-node* p vers q.

4.1. Politique de migration

Notre politique de migration doit préserver les invariants de distribution. Ces derniers pourraient être violés en choisissant mal le nœud à migrer ou en choisissant mal sa destination.

Un *p-node p* peut migrer un nœud *n* dont il est responsable vers un *p-node q* sous réserve que les invariants de distributions soient vérifiés après l'execution de la migration Les migrations suivantes ne respecteraient pas les invariants de distribution :

– si *p* est responsable d'exactement un nœud *n* : si *p* dplace *n*, *p* ne sera plus responsable d'une feuille (Inv_1)
– si *p* est responsable d'au moins deux nœuds : si *p* migre son *dernier* nœud, *p* ne sera plus responsable d'une feuille (Inv_1) si *p* migre un nœud *n* qui n'est ni son *premier* ni son *dernier* nœud, *p* ne sera plus responsable d'exactement un fils du père de *n* (Inv_2).

Cependant, si *p* migre son *premier* nœud *n* vers *q*, les invariants sont préservés si *q* est responsable d'un fils de *n*. Nous disons qu'un *v-node n* est "migrable" par un *p-node p* si :

Definition 1 ***Un* v-node** *n* ***est migrable par un* p-node** *p* ***si*** *p* ***est responsable d'au moins deux nœuds et*** *n* ***est le*** *premier* ***nœud de*** *p.*

Sur la figure 1a, seuls n_0, n_6 et n_9 sont migrables.

Nous allons maintenant voir comment choisir une destination valide, toujours de faon à préserver les invariants de distribution.

Soient *n* un nœud migrable par un *p-node p* et *q* une destination éventuelle de *n*. Les invariants ne seraient par respectés dans les situations suivantes :

– si *q* n'est responsable d'aucun voisin de *n* : par dfinition, *n* n'est pas une feuille donc *q* ne serait responsable d'aucun fils de *n* (Inv_2)
– si *q* est responsable du père de *n* : *q* serait responsable de deux fils du père de *n* (Inv_2)

Cependant, si *q* est responsable d'un fils de *n* et que *n* est migré vers *q*, alors *n* devient le *premier* nœud de *q*. Nous disons que *q* est une destination valide pour un nœud *n* si :

Definition 2 ***Un* p-node** *q* ***est une destination valide pour un* v-node** *n* ***si*** *q* ***est responsable d'un fils de*** *n.*

Sur la figure 1a, p_1 peut migrer n_0. Les destinations valides sont les *p-node* responsables des fils de n_0 : p_5 et p_7.

Le degr du DR-tree assure que le *p-node* responsable de la racine a toujours le choix entre 1 et M-1 migrations tandis que les autres *p-nodes* ont toujours le choix entre m-1 et M-1 migrations pour leur *premier* nœud.

4.2. Résolution des conflits de migration

La politique de migration assure la cohérence "locale" des migrations. Toutefois, deux migrations executées concurremment peuvent amener à une configuration invalide. Sur la figure 1a, p_1 peut décider d'excuter $p_1 \xrightarrow{n_0} p_5$ en même temps que p_5 décide d'excuter $p_5 \xrightarrow{n6} p6$. Ces deux migrations sont correctes au regard de la politique de migration. Cependant, après leur execution concurrente, p_5 est responsable des nœuds n_0 et n_7 ; l'invariant 2 n'est pas respecté car n_0 n'est pas une feuille et p_5 n'est responsable d'aucun de ses fils.

Quand un *v-node n* est migré d'un *p-node p* vers *q*, *p* et *q* doivent évidemment participer au protocole de migration. Afin d'éviter les conflits entre les migrations, les *p-nodes* responsables des voisins de *n* doivent également y participer. Le principe de notre protocole est que si un *p-node p* veut executer une migration vers *q*, il doit demander la permission aux autres *p-nodes* susceptibles d'éxecuter une migration conflictuelle. C'est une transaction distribuée dans laquelle tous les *p-nodes* impliqués doivent donner leur autorisation pour qu'elle soit validée. Autrement, la transaction est annulée.

La détection des *p-nodes* pouvant exécuter des migrations conflictuelles est très dépendante de la poli-

tique de migration. Avec celle que nous avons présentée, un *p-node* p ne peut recevoir de nœud que depuis son père. Il y a donc une certaine localité dans les conflits potentiels ; quand un pair p exécute une migration, il peut entrer en conflit avec son père ou un de ses fils dans le graphe de communication.
Un protocole simple permet d'éviter l'exécution de migrations concurrentes. Soit p un pair souhaitant exécuter une migration. Si p est la racine du graphe de communication, il exécute la migration. Sinon il demande l'autorisation à son père dans le graphe de communication ; s'il l'obtient il exécute la migration, autrement elle est annulée.

4.3. Planificateur de migrations

Le planificateur de migration définit quand les migrations doivent être exécutées, autrement dit quand le protocole de migration doit être utilisé. Il peut être invoqué périodiquement ou en réaction à une modification du DR-tree.
Dans le premier cas, le planificateur peut être implémenté comme un processus distinct. Il évalue périodiquement une fonction de coût (par exemple de charge) et quand un certain seuil est atteint, il vérifie s'il existe une migration qui peut améliorer la configuration et tente de l'exécuter. Dans le second cas, le planificateur est déclenché à certains points du cycle de vie de la structure. Par exemple en réaction au *split* d'un *v-node*, le planificateur peut être invoqué pour placer au mieux le nœud créé. Il est à noter que cette approche permet d'utiliser notre mécanisme pour faire du placement de nœud. Le point important est que le cycle de vie de la structure et celui du mécanisme de migration sont disjoints et peuvent donc tre combins.
Notre implémentation du planificateur est à nouveau conservatrice ; il tentera d'exécuter les migrations sous réserve que celles-ci améliorent à priori les performances du système.
La fonction de coût peut prendre en considération différentes métriques ; la bande passante, les capacités des machines etc...En pratique, son évaluation sera basée sur les informations qu'un *p-node* a de son voisinage. Les *p-nodes* peuvent rafraîchir cette information périodiquement ou en analysant le traffic circulant dans le DR-tree. Un exemple de fonction de coût est presenté dans la section 5.
L'absence de connaissance globale de la fonction de coût peut limiter l'efficacité de notre système ; certaines améliorations ne seront pas détectées. Supposons que sur la figure 1a p_2 a la meilleure bande passante ; idéalement il devrait être responsable des nœuds proches de la racine. Cependant si d'après l'évaluation de la fonction de coût de p_1 ni $p_1 \xrightarrow{n_0} p_5$ ni $p_1 \xrightarrow{n_0} p_7$ n'améliorent la configuration du système, p_1 ne migrera pas n_0. Or tant que p_1 est responsable de n_0, $p_1 \xrightarrow{n_1} p_2$ ne peut pas être exécutée. La connaissance limitée de la fonction de coût ne permettra donc pas p_2 de recevoir des nœuds même si cela améliorerait les performances du système.

5. Evaluation

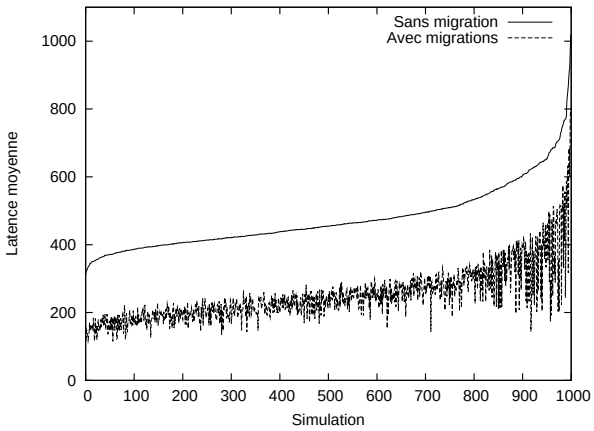
Nos expérimentations ont été menées sur un simulateur discret ad-hoc. Au début de chaque cycle, chaque *p-node* vérifie avec une certaine probabilité s'il peut améliorer les performances du systèmes en migrant son *premier* noeud. Pour ce faire le pair évalue la fonction de coût pour chaque migration possible.
Une migration $m{=}p \xrightarrow{n} q$ concerne l'ensemble des pairs responsables des voisins de n ; notons cet ensemble de pairs $concerned(m)$. Nous supposons que p connait la fonction de coût entre ses voisins du graphe de communication. Pour décider si m améliore les performances du système, p attribue un score à chaque migration :

$$score(m) = \sum_{r}^{concerned(m)} cost(p,r) - cost(q,r)$$

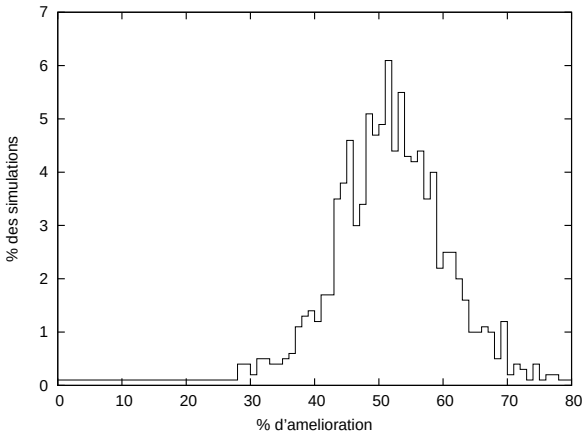
Si $score(m) > 0$ alors, d'après p, m peut améliorer les performances du système. p tentera d'exécuter la

migration avec le plus grand score s'il est strictement positif. Dans nos expérimentations, la fonction de coût est basée sur la latence entre les pairs.
1000 simulations ont été exécutées avec différents DR-tree en utilisant les paramètres suivants :
– 500 *p-nodes*
– un degré de m = 2 et M = 4
– chaque simulation dure 500 cycles
– à chaque cycle, la probabilité qu'un *p-node* essaie d'améliorer la configuration est de 1/10
Le DR-tree stocke des rectangles 2D de 50x50 répartis aléatoirement dans une carte de 1024x1024. Pour simuler la latence entre les *p-nodes*, nous nous sommes basés sur le Meridian data set [16]. Cette matrice de latence reflète le round-trip moyen entre 2500 pairs échantillonés sur internet. Pour nos simulations nous avons extrait une sous matrice de 500x500.

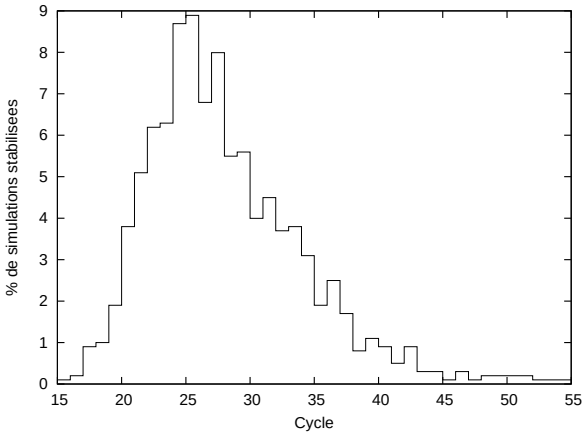
5.1. Impact des migrations sur la latence moyenne du graphe de communication



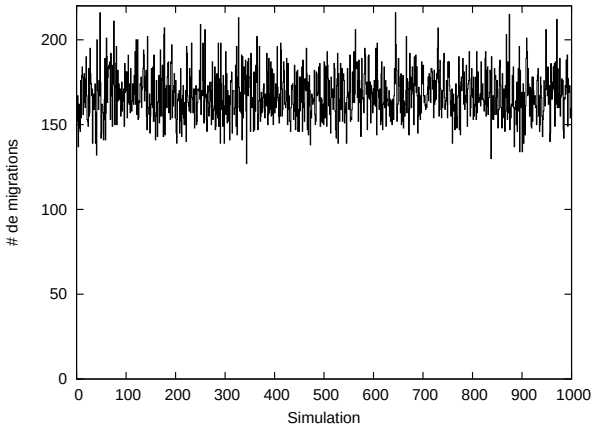
(a) Amélioration moyenne de la latence



(b) Distribution de l'amélioration moyenne de la latence



(c) Distribution du temps de stabilisation



(d) Nombre de migrations par simulation

min	max	moyenne	écart type
15	55	27.494	5.84209

(e) Statistiques du temps de stabilisation

min	max	moyenne	écart type
127	216	167.793	14.530

(f) Statistiques du nombre de migrations par simulation

FIGURE *3* – Mesures pour 1000 simulations

Nous avons d'abord étudié l'impact des migrations sur la latence moyenne du graphe de communication (i.e., la latence moyenne entre deux nœuds voisins dans le graphe). Cette latence a une influence directe sur les performances du DR-Tree.

La figure 3a montre le gain moyen de latence obtenu grace aux migrations. En abscisse, les simulations sont triées par latence moyenne croissante. L'ordonnée indique la latence moyenne en milli-secondes dans le graphe de communication après stabilisation du système, c'est à dire quand plus aucune migration n'est exécutée.

La latence moyenne sans migration -et donc sans tenir compte de l'hétérogénéité du réseau- est étroitement liée au placement de la racine du DR-tree et de son voisinage proche. Dans quelques rares cas, ce placement est bon (resp. mauvais) menant à une latence moyenne de 300ms (resp. 1000ms). Dans 80% des simulations, la latence moyenne est comprise entre 350ms et 600ms.

Cette figure montre également que les migrations améliorent clairement la latence moyenne. Les pics de la courbe s'expliquent de la même manière : dans un petit nombre de cas, les migrations exécutées sont nombreuses et efficaces (resp. peu nombreuses et peu efficaces) menant à une latence moyenne de 150ms (resp. 800ms). Dans la plupart des cas le gain avoisine 50%.

La figure 3b montre la distribution de ce gain. L'abscisse indique le pourcentage de gain obtenu pendant une simulation. L'ordonnée indique le pourcentage de simulations où le gain x a été atteint. La distribution est gaussienne : 78.7% des simulations atteignent un gain compris entre 40% et 60%.

La figure 3c montre la distribution du temps de stabilisation du système. L'abscisse indique le dernier cycle où une migration a été exécutée. L'ordonnée indique le pourcentage de simulations où la dernière migration a été exécutée au cycle x.

La distribution du temps de stabilisation est gaussienne : 92.8% des simulations convergent entre les 20 et 40 cycles. Même avec des *p-nodes* essayant "rarement" d'améliorer la configuration du système, la stabilisation est très rapide : alors que les simulations durent 500 cycles, 96.1% d'entre elles convergent en moins de 40 cycles (100% en moins de 55 cycles).

La figure 3d montre le nombre de migrations exécutées par simulation qui s'avère être relativement stable. En effet, cette quantité dépend principalement des propriétés structurelles de l'arbre, en particulier de son degré, et des latences entre les pairs. Or ces différents critères ne changent pas entre les simulations.

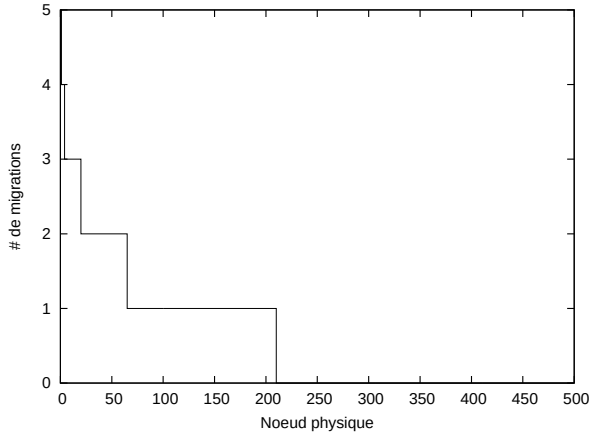
5.2. Analyse d'une simulation représentative

Les résultats de cette section se basent sur une simulation représentative avec un gain de latence de 50% (Figure 3b), un temps de stabilisation de 25 cycles (Figure 3c) et 173 migrations exécutées (Figure 3d).

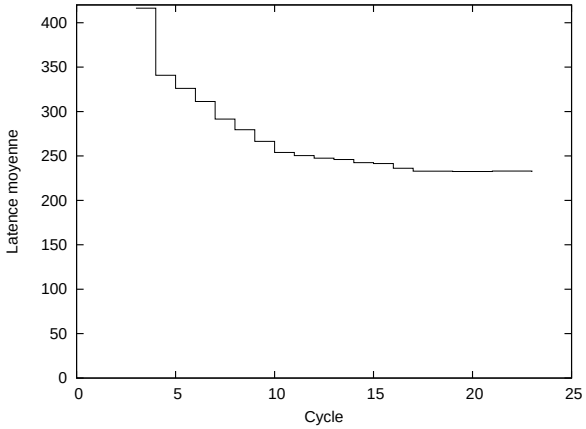
La figure 4a montre le nombre de migrations exécutées par chaque *p-node* durant la simulation de référence. Plus de la moitié des *p-node* ne participent à aucune migration et aucun *p-node* ne participe à plus de cinq migrations. La distribution du nombre de migrations par *p-node* est plutôt bien équilibrée puisqu'aucun *p-node* n'est surchargé et seulement 4% des *p-nodes* participent à plus de deux migrations.

La figure 4b montre l'évolution de la latence moyenne dans le graphe de communication au cours du temps. Tous les *p-nodes* démarrent leurs planificateurs au premier cycle de la simulation. Toutefois le protocole de migration nécessite quatre cycles pour valider ou annuler une migration ; la première migration est donc exécutée au quatrième cycle. Par ailleurs les *p-nodes* éloignés de la racine du graphe de communication ont plus de chance de voir leurs migrations refusées car leurs pères ont plus de chances d'être en train d'en exécuter. Plus un *p-node* est proche de la racine du graphe de communication, plus les migrations qu'il exécute ont un impact important sur la latence moyenne. Les autres migrations sont des ajustements de moindre importance ou permettent à terme de promouvoir des *p-nodes* très performants initialement éloignés de la racine du graphe de communication.

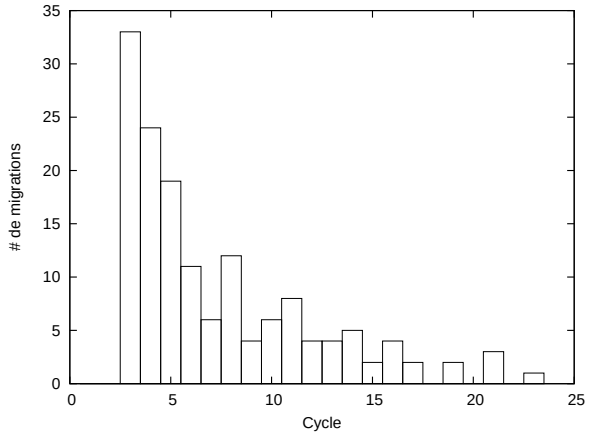
La figure 4c montre le nombre de migrations exécutées par cycle. L'abscisse indique le temps en cycle. L'ordonnée indique le nombre de migrations exécutées durant le cycle x. Ces résultat confirment l'analyse



(a) Nombre de migrations par pair



(b) Latence moyenne en fonction du temps



(c) Nombre de migrations par cycle

FIGURE *4* – Mesures sur une simulation représentative

de la figure 4b : comme tous les *p-nodes* démarrent leurs planificateurs au premier cycle, bon nombre d'entre eux exécutent leurs première migration au quatrième cycle. Comme tous les *p-nodes* sont présents dès le début de la simulation et qu'il n'y a aucun départ ni aucune arrivée, plus le temps avance, moins les *p-nodes* trouvent de migrations intéressantes, moins ils en exécutent.

6. Conclusion

Notre article analyse l'impact de la distribution de la structure logique du DR-tree sur le graphe de communication des nœuds physiques. Notre mécanisme de migration dynamique de noeuds permet de modifier le graphe de communication, et donc les performances du système, sans modifier la structure logique. Nos résultats expérimentaux montrent que même une implémentation naïve de notre architecture permet une amélioration substancielle des performances du système en terme de latence. Finalement, il est important de souligner que le concept de distribution est facilement généralisable à d'autres types de topologies et de structures.

Bibliographie

1. S. Baehni, P. T. Eugster, et R. Guerraoui. Data-aware multicast. In *DSN*, page 233, 2004.
2. S. Bianchi, A. K. Datta, P. Felber, et M. Gradinariu. Stabilizing peer-to-peer spatial filters. In *ICDCS 2007*, page 27, 2007.
3. Paris C. et Vana K. A topologically-aware overlay tree for efficient and low-latency media streaming. In *QSHINE*, pages 383–399, 2009.
4. R. Y. Wang C. Zhang, A. Krishnamurthy. Brushwood : Distributed trees in peer-to-peer systems. pages 47–57, 2005.
5. M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. I. T. Rowstron, et A. Singh. Splitstream : High-bandwidth content distribution in cooperative environments. In *IPTPS*, pages 292–303, 2003.
6. G. Chockler, R. Melamed, Y. Tock, et R. Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *PODC*, pages 109–118, 2007.
7. Y. Choi et D. Park. Mirinae : A peer-to-peer overlay network for large-scale content-based publish/subscribe systems. In *NOSSDAV*, pages 105–110, 2005.
8. P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, et A-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4) :341–374, 2003.
9. R. Guerraoui, S. B. Handurukande, K. Huguenin, A-M. Kermarrec, F. Le Fessant, et E. Riviere. Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. In *Peer-to-Peer Computing*, pages 12–22, 2006.
10. A. Gupta, O. D. Sahin, D. Agrawal, et A. E. Abbadi. Meghdoot : content-based publish/subscribe over p2p networks. In *Middleware*, pages 254–273, 2004.
11. A. Guttman. R-trees : a dynamic index structure for spatial searching. In *ACM SIGMOD*, pages 47–57, 1984.
12. N. J. A. Harvey et J. I. Munro. Deterministic skipnet. *Inf. Process. Lett.*, 90(4) :205–208, 2004.
13. H. V. Jagadish, B. C. Ooi, et Q. H. Vu. Baton : a balanced tree structure for peer-to-peer networks. In *VLDB*, pages 661–672, 2005.
14. H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, et A. Zhou. Vbi-tree : A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *ICDE*, page 34, 2006.
15. Y-J. Joung. Approaching neighbor proximity and load balance for range query in p2p networks. *Comput. Netw.*, 52(7) :1451–1472, 2008.
16. H. V. Madhyastha, T. Anderson, A. Krishnamurthy, N. Spring, et A. Venkataramani. A structural approach to latency prediction. In *IMC*, pages 99–104, 2006.
17. J. Pujol Ahullo, P. Garcia Lopez, et A. F. Gomez Skarmeta. Towards a lightweight content-based publish/subscribe services for peer-to-peer systems. *Int. J. Grid Util. Comput.*, 1(3) :239–251, 2009.
18. S. Ratnasamy, P. Francis, M. Handley, R. Karp, et S. Schenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.
19. A. Rowstron et P. Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.
20. W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, et A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *DEBS*, pages 1–8, 2003.